有理数行列のFrobenius標準形のモジュラー計算法

AUTHOR(S):

森継, 修一

CITATION:

森継, 修一. 有理数行列のFrobenius標準形のモジュラー計算法. 数理解析研究所講究録 2003, 1335: 33-40

ISSUE DATE:

2003-07

URL:

http://hdl.handle.net/2433/43336

数理解析研究所講究録 1335 巻 2003 年 33-40

有理数行列のFrobenius標準形のモジュラー計算法

森継 修一

SHUICHI MORITSUGU

筑波大学 図書館情報学系

INSTITUTE OF LIBRARY AND INFORMATION SCIENCE, UNIVERSITY OF TSUKUBA*

1 はじめに

行列のFrobenius標準形（有理標準形）は、体の拡大を必要とするJordan標準形と異なり、行列要素間の四則演算（有理演算）のみで計算が可能である。さらにFrobenius標準形は、もとの行列の特性多項式・最小多項式、固有値の代数的・幾何学的重複度や対応する（一般）固有ベクトルの構成などの完全な情報をJordan標準形と同等に含んでおり [13][14][16]、数式処理による行列計算のアルゴリズムを考えるには、拡大体上の計算が必要なJordan標準形よりもFrobenius標準形を基礎とする方が適している。本研究では、固有ベクトルの計算 [16] や固有値法による連立代数方程式の解法 [19][15] への応用を想定して、与えられた行列 A のFrobenius標準形 F と変換行列 S $(AS = SF)$ の両方を求めることを目的とする。これまでは、整数行列に限ってモジュラー法の適用を検討してきた [12] が、今回、対象を有理数要素の行列に拡張する。

2 行列のFrobenius標準形

以下の議論は任意の体の元を要素とする行列に対して成り立つが、具体的には、厳密な数値を表現する有理数にとるものとする。すなわち、$A = [a_{ij}]$, $a_{ij} \in \mathbf{Q}$ とおく。

定義 1 (コンパニオン行列)
次の $n \times n$ 正方行列

$$C = \begin{bmatrix} 0 & 1 & & & \\ 0 & 0 & \ddots & & O \\ \vdots & \vdots & \ddots & \ddots & \\ 0 & 0 & \cdots & 0 & 1 \\ c_0 & c_1 & \cdots & c_{n-2} & c_{n-1} \end{bmatrix} \tag{1}$$

は、多項式 $f(x) = x^n - c_{n-1}x^{n-1} - \cdots - c_1 x - c_0$ に随伴するコンパニオン行列と呼ばれる。特に、1次多項式 $f(x) = x - c_0$ のコンパニオン行列は 1×1 行列 $[c_0]$ とする。

補題 2
行列 C (1) の特性多項式 $\varphi_C(x)$ と最小多項式 $\phi_C(x)$ は、随伴多項式 $f(x) = x^n - c_{n-1}x^{n-1} - \cdots - c_1 x - c_0$ に一致する。

*moritsug@slis.tsukuba.ac.jp

定理 3 (Frobenius 標準形)

(i) 任意の $n \times n$ 正方行列 A は、適当な正則行列 S により次の形のブロック対角行列に相似変換され、これを A の Frobenius（または有理）標準形という。

$$F = S^{-1}AS = C_1 \oplus C_2 \oplus \cdots \oplus C_t. \tag{2}$$

各ブロック行列 C_i $(i = 1, \cdots, t)$ は、$d_i \times d_i$ コンパニオン行列 (1) であり、C_{i+1} の随伴多項式 $\varphi_{i+1}(x)$ は、C_i の随伴多項式 $\varphi_i(x)$ を割り切る $(i = 1, \ldots, t-1)$。

(ii) 与えられた行列 A に対して、(2) の形の分解は常に存在し、かつ Frobenius 標準形 F は一意的に決まる。（相似変換行列 S は一意ではない。）さらに、A の最小多項式 $\phi_A(x)$ は $\varphi_1(x)$ に一致し、A の特性多項式 $\varphi_A(x)$ は $\varphi_1(x) \cdot \varphi_2(x) \cdots \varphi_t(x)$ で与えられる。

行列を (2) のようなブロック対角形に変換して特性多項式を求める方法が Danilevskii 法 [3][4] として古くから知られているが、定理 3 の厳密な意味での有理標準形は、「ベクトル空間の巡回分解定理」の構成的証明 [5][7][9] によって与えられる。この証明に直接に基づく算法は、関係式 $F = S^{-1}AS$ における変換行列 S を先に求めていく形で表現され、数式処理システム Maple[2] の組込関数 frobenius は、この方針で実装されていると思われる。これに対し本研究では、[9] を同著者が行列の基本変形の表現に翻訳した形のアルゴリズム [10] を採用する。この場合は、標準形 F を求めることに付随して変換行列 S も求められ、その計算量は $O(n^3 \log n)$ のクラス—$O(n^3)$ が行列積などの基本演算に対応、$O(\log n)$ は標準形 (2) におけるブロックの個数 t に相当—に属する [1] と考えられる。この算法では、次の基本変形操作を組み合わせて、Gauss 消去法に類似の消去計算を実行することにより、与えられた行列 A を標準形 F に変換していく。

定義 4 (基本変形)

任意の $n \times n$ 正方行列 A に対する次の 3 つの操作を基本変形と呼ぶ。

$op1(k, \ell)$: A の第 k 行と第 ℓ 行を入れ換え、続いて第 k 列と第 ℓ 列を入れ換える。

$op2(k, c)$: A の第 k 行を c^{-1} 倍し、続いて第 k 列を c 倍する。

$op3(k, \ell, c)$: A の第 k 行に第 ℓ 行の c 倍を加え、続いて第 ℓ 列から第 k 列の c 倍を引く。

これらの基本変形は行列の相似変換 $A \mapsto S^{-1}AS$ で表現され、行に対する操作が左の変換行列 S^{-1} に、列に対する操作が右の変換行列 S に対応している。消去に必要な相似変換を $\cdots S_3^{-1} \left(S_2^{-1} \left(S_1^{-1} A S_1 \right) S_2 \right) S_3 \cdots$ のように順次適用していくと、最終的に式 (2) における F, S, S^{-1} が得られるが、基本変形操作の定義より、この過程で必要となるのは有理数同士の四則演算のみであることに注意する。（ただし、行列 A の要素をすべて整数にとった場合は、その Frobenius 標準形の各要素も整数となる。）

3 整数行列に対するモジュラー計算アルゴリズム

本章では、整数行列 A に対して $AS = SF$ をみたす F をモジュラー計算で求め、その後、S の各要素が（できるだけ簡潔な）整数で得られるようにアルゴリズムを構成する。基本変形 $op2(k, c)$ には「c^{-1} 倍する」という除算が含まれるが、Euclid 互除法により、$cs + pt = 1$ をみたす s, t を求めれば、$s \equiv c^{-1} \pmod{p}$ となり、この除算は乗算の形で実現される。したがって、基本変形における有理演算はすべて素数 p を法として実行可能であり、$\mathbf{Z}_p$ での Frobenius 標準形 $A_p S_p = S_p F_p$ が $\mathbf{Z}$ 上と同じプログラムで求められる。

複数の素数を法として計算した結果 $F_{p_1}, F_{p_2}, \ldots$ から $\mathbf{Z}$ 上での F を復元するには Chinese Remainder Theorem を適用する。（後述のように、S を CRT で復元するのは非効率のため、S_{p_i} は保存しない。）

定理 5 (CRT：法が 2 つの場合)
m_1, m_2 が互いに素な整数のとき、連立合同式の解は、$m_1 s + m_2 t = 1$ を満たす 1 組の整数 s, t を用いて、次の式で与えられる。

$$\begin{cases} x \equiv a_1 & (\mathrm{mod}\ m_1) \\ x \equiv a_2 & (\mathrm{mod}\ m_2) \end{cases} \quad \Longrightarrow \quad x \equiv a_1 m_2 t + a_2 m_1 s \quad (\mathrm{mod}\ m_1 m_2) \tag{3}$$

したがって、素数 p_1, p_2, $p_3, \ldots$ に対して、法を p_1, $p_1 p_2$, $p_1 p_2 p_3, \ldots$ と上げていくには、

$$\begin{cases} x \equiv a_{k-1} & (\mathrm{mod}\ p_1 \cdots p_{k-1}) \\ x \equiv a_k & (\mathrm{mod}\ p_k) \end{cases} \quad (k = 2,\ 3, \ldots) \tag{4}$$

に対して、定理 5 を繰り返し適用すればよい（Newton の補間公式)。標準形 F に対しては、0, 1 以外の値を持つ n 個の要素 f_{ij} について、個別に計算することになる。

3.1 unlucky な素数の発見と回避

定義 6 (unlucky な素数)
$\mathbf{Z}$ 上で求めた標準形 $AS = SF$ および、$\mathbf{Z}_p$ での標準形 $A_p S_p = S_p F_p$ とにおいて、$F \neq F_p \pmod p$ または $S \neq S_p \pmod p$ となる場合、素数 p は unlucky であるという。

補題 7 (Howell[8])
lucky な素数 p を法とする場合の pivot 選択のパターンは、$\mathbf{Z}$ 上における pivot 選択パターンと一致する。

この補題によれば、消去計算の過程における pivot 選択の履歴を記録し比較することにより lucky/unlucky が識別できる。これは「unlucky な素数 p が偶然 pivot 要素を割り切って軸位置に 0 がくると、本来起きないはずの行交換・列交換 $op1(k, \ell)$ が起きる」ことを意味する。各ステップにおける pivot の素因数は有限個なので、特定の行列に対する unlucky な素数は全体として有限個しか存在しない。ただし、「$\mathbf{Z}$ 上での pivot 選択のパターン」は未知なので、あくまでも推定に基づくが、実用上その推定を誤る確率はごく小さい [12]。

3.2 変換行列の構成法

「相似変換を順次適用して行列の消去を行なう」という算法を図式的に表せば、

$$\boxed{E}\boxed{A}\boxed{E} \quad \longrightarrow \quad \boxed{S^{-1}}\boxed{F}\boxed{S} \tag{5}$$

となる。ここで、左側の単位行列には行操作を、右側の単位行列には列操作を順次適用する。本研究では右側の S だけを求めることにし、その要素が整数になるよう列操作の表現行列 S_k を定めているが、消去の過程 $S = S_1 S_2 \cdots$ において相似変換のための列操作が累積していくと、一般に S の要素は巨大な整数に成長してしまう。モジュラー計算の結果 $S_{p_1}, S_{p_2}, \ldots$ から CRT で S を復元するような実装では、S の要素の成長に伴って法 p_i の個数を増やさなければならないため、計算速度の向上には十分つながらなかった [17]。

一方で、Frobenius 標準形の定義より、相似変換行列 S は一意ではないため、$AS = SF$ をみたす正則なものが 1 つ見つかれば十分である。これまでの研究から、「標準形 F だけを CRT で先に求め、その後 $AS = SF$ をみたす正則行列 S（しかもできるだけ簡潔な要素を持つもの）を構成する」アルゴリズムが最も効率的であることが分かってきた [12]。関係式 $AS = SF$ の両辺を列ベクトル毎に比較することにより、与えられた A と計算済みの F から S を求めるには、以下の算法を適用すればよい。計算量としては、$n \times n$ の連立 1 次方程式を、(ブロックの個数 -1) 回だけ解く必要があるので、この部分も $O(n^3 \log n)$ である。

アルゴリズム 1 (変換行列の再構成の算法)

% 入力：整数要素の $n \times n$ 行列 A とその Frobenius 標準形 $F = C_1 \oplus C_2 \oplus \cdots \oplus C_t$

% 各 C_k の随伴多項式を $\varphi_k(x) = x^{d_k} - c_{k,d_k-1}x^{d_k-1} - \cdots - c_{k,1}x - c_{k,0}$ とする。

% 出力：$AS = SF$ をみたす行列 $S = \left[s_{1,1} \middle| \cdots \middle| s_{1,d_1} \middle| \cdots \middle| s_{t,1} \middle| \cdots \middle| s_{t,d_t} \right]$

for $k = 1$ to t do

Compute s_{k,d_k} by solving $\varphi_k(A)s_{k,d_k} = \mathbf{0}$;

for $j = d_k - 1$ downto 1 do $s_{k,j} = As_{k,j+1} - c_{k,j}s_{k,d_k}$;

以上をまとめると、次のようなアルゴリズムになる。ここで、行列 A に対して $AS = SF$ なる F, S を求めるサブプログラムは、すでに出来ているものとする。$F^{(k-1)} = F^{(k)}$ となった時点で $\varphi(A) = O$ を確かめているので、CRT の繰り返しが終了すれば、正しい Frobenius 標準形 F が得られていることになる。

アルゴリズム 2 (Frobenius 標準形のモジュラー計算＋変換行列の再構成)

% 入力：整数要素の $n \times n$ 行列 A　　ある程度大きな素数のリスト $\{p_1, p_2, \ldots, p_s\}$

% 計算過程での $F^{(k)}$ は mod $p_1 \cdots p_k$ （ただし unlucky と判定された p_i を除く）

% での値を表す。S_{p_k} は保存しない。

% 出力：A の Frobenius 標準形 F と相似変換行列 S　（$AS = SF$）

Compute F_{p_i} s.t. $A_{p_i}S_{p_i} \equiv S_{p_i}F_{p_i}$ (mod p_i $i = 1, 2, 3$);

Presume the correct pivoting pattern using the above 3 results;

% ここでは、p_1, p_3 のパターンが一致したとして、これを正しいパターンと仮定し、

% p_2 は異なるパターンをたどったので unlucky と判定したとする。

Construct $F^{(3)}$ from F_{p_1} and F_{p_3} by CRT;

$k := 3$; $count := 1$;

loop : Do until $(F^{(k-1)} = F^{(k)})$

$k := k + 1$;

Compute F_{p_k} s.t. $A_{p_k}S_{p_k} \equiv S_{p_k}F_{p_k}$ (mod p_k);

If p_k does not yield the presumed pivoting pattern

then { $count := count + 1$; If $count = 10$ then STOP; }

else Construct $F^{(k)}$ from $F^{(k-1)}$ and F_{p_k} by CRT;

If $\varphi(A) \neq O$ then goto loop; % $\varphi(x)$ = min.pol. of $F^{(k)}$

Construct S from $A, F^{(k)}$ by Algorithm-1 (over $\mathbf{Z}$);

While rank$(S) < n$ (over $\mathbf{Z}$) do reconstruct S by Algorithm-1;

Return $(F^{(k)}, S)$

3.3 実装環境

- 数式処理システム Reduce3.7[6](Windows 版)+RLISP '88[11]
  多項式および有理数計算：algebraic mode　　$\mathbf{Z}$ および $\mathbf{Z}_p$ 上の計算：symbolic mode
- CPU：Pentium4 (2GHz)　　使用メモリ：1GB(実装 1.5GB)
- 2^{31} 未満の素数リスト $\{p_1, p_2, \ldots, p_{1000}\}$ = {2147483647, 2147483629, ..., 2147462143}
- テスト行列：ランダムな有理数を係数にもつ特性多項式→結果として想定される有理標準形
  →相似変換によりランダムな配列の行列（要素は $n = 100$ で最長 200 桁/200 桁程度）
- 比較・検証用：Maple7(Windows 版)

4 有理数行列に対する計算法

4.1 整数上で計算を進める方法

有理数行列 A に対して、要素の共通分母 k を括り出し、$\tilde{A} = kA \quad (a_{ij} \in \mathbf{Q}, \quad k,\ \tilde{a}_{ij} \in \mathbf{Z})$ とおいて、可能な限り整数上で計算を進める方法が考えられる。すなわち、$\tilde{A}$ に対してアルゴリズム 2 を適用して

$$\tilde{S}^{-1}\tilde{A}\tilde{S} = \begin{bmatrix} & 1 & & \\ & & 1 & \\ & & & 1 \\ p & q & r & s \end{bmatrix} = \tilde{F} \qquad (p,q,r,s \in \mathbf{Z}) \tag{6}$$

をみたす $\tilde{F}, \tilde{S}$ を $\mathbf{Z}$ 上で求めたとする（例として、4×4 でブロック 1 個の場合）。これに対し、

$$V = \begin{bmatrix} 1 & & & \\ & k & & \\ & & k^2 & \\ & & & k^3 \end{bmatrix} \tag{7}$$

とおいて $\tilde{F}/k = \tilde{S}^{-1}\left(\tilde{A}/k\right)S$ を相似変換し、

$$V^{-1}\left(\tilde{F}/k\right)V = \begin{bmatrix} & 1 & & \\ & & 1 & \\ & & & 1 \\ p/k^4 & q/k^3 & r/k^2 & s/k \end{bmatrix} =: F \tag{8}$$

とおくと、この F が $\tilde{A}/k = A$ の Frobenius 標準形である。（$\tilde{F} = C_1 \oplus C_2 \oplus \cdots \oplus C_t$ のときは、各ブロックのサイズに対応させた $V = V_1 \oplus V_2 \oplus \cdots \oplus V_t$ を作る。）このとき $(\tilde{S}V)^{-1}A(\tilde{S}V) = F$ より $S := \tilde{S}V$ が $AS = SF$ をみたす相似変換行列となる。（$a_{ij} \in \mathbf{Q}$ にもかかわらず $s_{ij} \in \mathbf{Z}$ である。）

4.2 整数・有理数変換を利用する方法

アルゴリズム 3 (有理数の復元　Wang, 1981)

% 入力：整数 c　法 m　$(0 < u < m)$　$\tilde{m} := \sqrt{m/2}$

% 出力：$a/b \equiv c \pmod m$　$-\tilde{m} < a < \tilde{m}, \quad 0 < b < \tilde{m}$

(R1) $u := (1, 0, m)$;　$v := (0, 1, c)$;

(R2) while $v_3 \geq \tilde{m}$ do $\{q := u_3/v_3;\quad r := u - qv;\quad u := v;\quad v := r\}$;

(R3) if abs$(v_2) \geq \tilde{m}$ then ERROR;

(R4) $a :=$ sign$(v_2)v_3$;　$b :=$ abs(v_2);

(R5) return $((a, b))$;

このアルゴリズムにより モジュラー計算の結果から A の標準形 F (over $\mathbf{Q}$) が直接復元できる。ただし、A の要素 a_{ij} の分母を割り切る p は unlucky として最初から除外しておく。この時点で S は未知である。

$$S^{-1}AS = \begin{bmatrix} & 1 & & \\ & & 1 & \\ & & & 1 \\ p' & q' & r' & s' \end{bmatrix} =: F \qquad (p',q',r',s' \in \mathbf{Q}) \tag{9}$$

この F に対して 先の (7) と同じ V により (8) と逆の相似変換を行なうと、

$$V(kF)V^{-1} = \begin{bmatrix} & 1 & & \\ & & 1 & \\ & & & 1 \\ k^4p' & k^3q' & k^2r' & ks' \end{bmatrix} =: \tilde{F} \tag{10}$$

として $\tilde{A}$ に対する $\mathbf{Z}$ 上の標準形が得られる。引き続いて、$\tilde{A}\tilde{S} = \tilde{S}\tilde{F}$ をみたす $\tilde{S}$ を再構成し、$S := \tilde{S}V$ とすれば $AS = SF$ をみたす相似変換行列（$s_{ij} \in \mathbf{Z}$）が得られる。

4.3 アルゴリズムの比較

算法 Rat　[10] にしたがい、有理数上で直接 F, S $(AS = SF)$ を計算する。

算法 Zmod

(1) $\tilde{A} := kA$ として共通分母を括り出す。

(2) モジュラー計算 + CRT で $\tilde{F}$ (over $\mathbf{Z}$) を求める。

(3) $\tilde{A}, \tilde{F}$ から $\tilde{S}$ (over $\mathbf{Z}$) を再構成する。　[注] ステップ (2)(3) がアルゴリズム 2 に相当

(4) $\tilde{F}$ を F (over $\mathbf{Q}$) に変換し、$S := \tilde{S}V$ を計算する。

算法 Qmod

(1) モジュラー計算 + CRT + 整数・有理数変換で F (over $\mathbf{Q}$) を求める。

(2) $\tilde{A} := kA$ に対する $\tilde{F}$ (over $\mathbf{Z}$) に変換する。

(3) $\tilde{A}, \tilde{F}$ から $\tilde{S}$ (over $\mathbf{Z}$) を再構成する。

(4) $S := \tilde{S}V$ を計算する。

5 実験的評価と今後の課題

表 1 は、Nonderogatory な（＝コンパニオンブロック 1 つからなる Frobenius 標準形を持つ）行列に対する計算結果（$\#p_i$ は法として用いた素数の個数）である。ここで、

- 上段：対角化不可能な行列（最小多項式が平方因子を含む）
- 下段：対角化可能な行列（最小多項式が無平方）

という違いがあり、これが計算時間に大きく影響を与えている。この原因は、「参考」として示したとおり、算法 Rat における変換行列 S の要素の桁数にあって、対角化不可能な行列（上段）に対しては、算法 Rat が特異的に速くなっていると見ることができる。すなわち、行列のサイズ n だけでは計算時間が決まらない不安定さを算法 Rat は持っている。結果として、上段のケースでは算法 Zmod は使い物にならないが、算法 Qmod は上段・下段の両ケースとも、算法 Rat に対して優っている。

表 2 は、Derogatory な（＝ Frobenius 標準形が 2 個以上のブロックに分かれる）行列に対する計算結果である。全体として、凸凹はあるものの Qmod で 80%以上の時間短縮になっており、効果が確認された。ここで使用した行列は、すべて対角化可能（最小多項式が無平方）なものであるが、対角化不可能な行列を対象としても、表 1 のような違いは見られなかった。

計算結果の検証および計算時間の比較のため、Maple7 の組込関数 frobenius での計算も行なった。アルゴリズムの詳細は不明であるが、変換行列 P（本稿の記法では、S^{-T} に相当）の要素の大きさによって計算時間が大きく変動するため、算法 Rat と同じような傾向が見られた。

表 1: CPU-Time(sec) for rational matrices I

	Rat	Zmod			Qmod			Maple	
n	T_R	$\#p_i$	T_Z	T_Z/T_R	$\#p_i$	T_Q	T_Q/T_R	T_M	T_Q/T_M
30	19.20	171	13.34	0.695	13	2.48	0.129	108.36	0.023
35	37.20	234	29.53	0.794	15	5.36	0.144	238.91	0.022
40	66.45	285	54.08	0.814	16	9.56	0.144	481.57	0.020
45	110.61	376	106.97	0.967	18	18.33	0.166	871.97	0.021
50	168.42	465	189.12	1.123	20	31.88	0.189	1427.22	0.022
100	3845.44	>1000	(failure)		39	1170.11	0.304	51088.66	0.023
30	23.22	162	12.50	0.538	12	2.28	0.098	135.86	0.017
35	55.34	216	26.83	0.485	14	4.83	0.087	371.56	0.013
40	152.30	320	63.08	0.414	18	11.09	0.073	1152.73	0.010
45	330.14	417	123.64	0.375	20	21.22	0.064	2635.40	0.008
50	696.86	530	228.25	0.328	23	37.70	0.054	5897.08	0.006
100	42331.16	>1000	(failure)		34	1067.83	0.025	(not tried)	

参考：$n = 50$ に対する s_{ij} の最長桁数

	算法 Rat	算法 Zmod	算法 Qmod
上段	2113 桁/2102 桁	4318 桁	4318 桁
下段	5023 桁/5024 桁	4899 桁	4899 桁

以上より、実験対象の範囲では、Zmod よりも断然 Qmod を採用するべきである。また、必要な法 p_i の個数の比較から、より桁数の多い有理数を要素とする行列の場合、Qmod がより優位となることは明らかである。整数・有理数変換については、アルゴリズム 3 のような古典的算法に対して、高速アルゴリズム [18] の研究が進められている。今回の実験で復元された有理数は高々200 桁／200 桁程度で、この部分が全体の計算時間に対して占める割合はごく小さく、また、現在の環境では高速アルゴリズムの本格的な実装は困難であるが、今後、適用が検討されるべきであろう。

参 考 文 献

[1] Augot, D. and Camion, P.: On the Computation of Minimal Polynomials, Cyclic Vectors, and Frobenius Forms, *Linear Algebra and its Applications*, **260**, 1997, 61–94.

[2] Char, B. W., et al.: *Maple V Library Reference Manual*, Springer, N.Y., 1991.

[3] Danilevskii, A. M.: On the numerical solution of the secular equation, *Mat.Sb.*, **2**(1), 1937, 169–172. (in Russian).

[4] ファジェーエフ, ファジェーエバ: 線型代数の計算法 (上), 産業図書, 東京, 1970.

[5] Gantmacher, F. R.: *The Theory of Matrices (2nd ed.)*, **1**, Chelsea, N.Y., 1959.

[6] Hearn, A. C.: *Reduce User's Manual (Ver. 3.7)*, RAND Corp., Santa Monica, 1999.

[7] Hoffman, K. and Kunze, R.: *Linear Algebra (2nd ed.)*, Prentice-Hall, New Jersey, 1971.

表 2: CPU-Time(sec) for rational matrices II

	Rat	Zmod			Qmod			Maple	
n (Structure)	T_R	$\#p_i$	T_Z	T_Z/T_R	$\#p_i$	T_Q	T_Q/T_R	T_M	T_Q/T_M
30 (15+15)	18.23	38	4.31	0.237	7	2.34	0.129	146.25	0.016
35 (20+15)	45.67	75	13.28	0.291	9	6.06	0.133	216.72	0.028
40 (25+15)	115.50	117	31.50	0.273	11	13.38	0.116	632.09	0.021
45 (30+15)	263.70	164	65.86	0.250	13	28.22	0.107	1579.33	0.018
.....	.....	.....			.....			.....	
50 (20+15+15)	128.24	75	38.92	0.304	9	19.47	0.152	866.04	0.022
55 (20+20+15)	351.95	75	63.78	0.181	9	36.27	0.103	2633.78	0.014
60 (25+20+15)	522.86	117	128.56	0.246	11	69.61	0.133	4350.91	0.016
65 (25+25+15)	1386.89	117	191.43	0.138	11	114.97	0.083	13083.23	0.009
70 (30+25+15)	1573.34	164	336.09	0.214	13	198.11	0.126	14743.82	0.013

参考：$n = 70$ に対する s_{ij} の最長桁数

算法 Rat	算法 Zmod	算法 Qmod
2601 桁/2602 桁	1569 桁	1570 桁

[8] Howell, J. A.: An Algorithm for the Exact Reduction of a Matrix to Frobenius Form Using Modular Arithmetic. I & II, *Math.Comp.*, **27**(124), 1973, 887–920.

[9] 伊理正夫, 韓太舜: 線形代数 - 行列とその標準形, 教育出版, 東京, 1977.

[10] 韓太舜, 伊理正夫: ジョルダン標準形, 東京大学出版会, 東京, 1982.

[11] Marti, J.: *RLISP '88: An Evolutionary Approach to Program Design and Reuse*, World Scientific, Singapore, 1993.

[12] 森継修一: 整数行列の Frobenius 標準形のモジュラー計算法, *J.JSSAC*(日本数式処理学会誌), 2003. (採録決定).

[13] 森継修一, 栗山和子: 行列の Jordan 標準形の数式処理による厳密計算法, 日本応用数理学会論文誌, **2**(1), 1992, 91–103.

[14] Moritsugu, S. and Kuriyama, K.: On Multiple Zeros of Systems of Algebraic Equations, *ISSAC 99*, ACM, N.Y., 1999, 23–30.

[15] Moritsugu, S. and Kuriyama, K.: A Linear Algebra Method for Solving Systems of Algebraic Equations, *J.JSSAC*(日本数式処理学会誌), **7**(4), 2000, 2–22.

[16] 森継修一, 栗山和子: 行列の固有値・固有ベクトル・一般固有ベクトルの数式処理による記号的計算法, 日本応用数理学会論文誌, **11**(2), 2001, 103–120.

[17] 森継修一, 栗山和子: 整数行列の Frobenius 標準形のモジュラー計算法, 京都大学数理解析研究所講究録, **1199**, 2001, 220–227.

[18] 佐々木建昭, 高橋善徳, 杉本卓也: 大整数に対する整数・有理数変換について, 京都大学数理解析研究所講究録, **1295**, 2002, 80–86.

[19] 竹島卓, 横山和弘: 連立代数方程式の一解法 - 剰余環上の線形写像の固有ベクトルの利用, 数式処理通信, **6**(4), 1990, 27–36.